Two-phase incremental kernel PCA for learning massive or online datasets

Zhao, Feng ; Rekik, Islem; Lee, Seong-Whan; Liu, Jing ; Zhang, Junying ; Shen, Dinggang

Published in:
Complexity

DOI:
10.1155/2019/5937274

Publication date:
2019

Hindawi
Complexity
Volume 2019, Article ID 5937274, 17 pages
https://doi.org/10.1155/2019/5937274

Research Article

Two-Phase Incremental Kernel PCA for Learning Massive or Online Datasets

Feng Zhao,[1,2] **Islem Rekik**,[3,4] **Seong-Whan Lee**,[5] **Jing Liu**,[6] **Junying Zhang**,[7] **and Dinggang Shen**[5,8]

[1]*School of Computer Science and Technology, Shandong Technology and Business University, Yantai, China*
[2]*Shandong Co-Innovation Center of Future Intelligent Computing, Yantai, China*
[3]*BASIRA Lab, Faculty of Computer and Informatics, Istanbul Technical University, Istanbul, Turkey*
[4]*School of Science and Engineering, Computing, University of Dundee, UK*
[5]*Department of Brain and Cognitive Engineering, Korea University, Seoul 02841, Republic of Korea*
[6]*School of Electronic Engineering, Xian University of Posts and Telecommunications, Xi'an, China*
[7]*School of Computer Science and Engineering, Xidian University, Xi'an, China*
[8]*Department of Radiology and BRIC, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA*

Correspondence should be addressed to Dinggang Shen; dgshen@med.unc.edu

Received 2 October 2018; Revised 17 December 2018; Accepted 8 January 2019; Published 11 February 2019

Guest Editor: Jose Garcia-Rodriguez

As a powerful nonlinear feature extractor, kernel principal component analysis (KPCA) has been widely adopted in many machine learning applications. However, KPCA is usually performed in a batch mode, leading to some potential problems when handling massive or online datasets. To overcome this drawback of KPCA, in this paper, we propose a two-phase incremental KPCA (TP-IKPCA) algorithm which can incorporate data into KPCA in an incremental fashion. In the first phase, an incremental algorithm is developed to explicitly express the data in the kernel space. In the second phase, we extend an incremental principal component analysis (IPCA) to estimate the kernel principal components. Extensive experimental results on both synthesized and real datasets showed that the proposed TP-IKPCA produces similar principal components as conventional batch-based KPCA but is computationally faster than KPCA and its several incremental variants. Therefore, our algorithm can be applied to massive or online datasets where the batch method is not available.

1. Introduction

As a conventional linear subspace analysis method, principal component analysis (PCA) can only produce linear subspace feature extractors [1], which are unsuitable for highly complex and nonlinear data distributions. In contrast, as a nonlinear extension of PCA, kernel principal component analysis (KPCA) [2] can capture the higher-order statistical information contained in data, thus producing nonlinear subspaces for better feature extraction performance. This has propelled the use of KPCA in a wide range of applications such as pattern recognition, statistical analysis, image processing, and so on [3–8]. Basically, KPCA firstly projects all samples from the input space into a kernel space using nonlinear mapping and then extracts the principal components (PCs) in the kernel space. In practice, such nonlinear mapping is performed implicitly via the "kernel trick", where an appropriately chosen kernel function is used to evaluate the dot products of mapped samples without having to explicitly carry out the mapping. As a result, the extracted kernel principal component (KPC) of the mapped data is nonlinear with respect to the original input space.

Standard KPCA has some drawbacks which limit its practical applications when handling big or online datasets. *Firstly*, in the training stage, KPCA needs to store and compute the eigenvectors of a $N \times N$ kernel matrix, where N is the total number of samples. This computation results in a space complexity of $O(N^2)$ and a time complexity of $O(N^3)$, thus rendering the evaluation of KPCA on large-scale datasets very time-consuming. *Secondly*, in the testing

stage, the resulting kernel principal components have to be defined implicitly by the linear expression of the training data, and thus all the training data must be saved after training. For a massive dataset, this translates into high costs for storage resources and increases the computational burden during the utilization of kernel principal components (KPCs). Furthermore, KPCA is impractical for many real-world applications where online samples are progressively collected since it is used in a batch manner. This implies that each time new data arrive, KPCA has to be conducted from scratch.

To overcome these limitations, many promising methods have been proposed in the past few years. These methods can be grouped into two classes. The first class is the batch-based modeling method, which requires that all training data is available for estimating KPCs. Rosipal and Girolami proposed an EM algorithm for reducing the computational cost of KPCA [9]. However, the convergence behavior of the EM algorithm to KPCA cannot be guaranteed in theory. In [6], the kernel Hebbian algorithm (KHA) was proposed as an iterative variant of KPCA algorithm. By kernelizing the generalized Hebbian algorithm (GHA), KHA computes KPCA without storing the kernel matrix, such that large-scale datasets with high dimensionality can be processed. Nonetheless, KHA has a scalar gain parameter which is either held constant or decreased according to a predetermined annealing schedule, leading to slow convergence during the training stage. To improve the convergence of KHA, gain adaptation methods were developed by providing a separate gain for each eigenvector estimate [10]. An improved version of KPCA was proposed based on the eigenvalue decomposition of a symmetric matrix [11], where datasets are divided into multiple subsets, each of which is processed separately. One of the major drawbacks of this approach is that it requires storing the kernel matrix, which means that the space complexity could be extremely large for large-scale dataset. Another variant of conventional KPCA is greedy KPCA [12, 13], which was employed to approximate the KPCs by a prior filtering of the training data. However, prior filtering of the training data could be computationally expensive. Overall, compared with standard KPCA, these batch-based modeling methods can potentially reduce the time or space complexity to some degree. Unfortunately, such methods cannot handle online data.

The second class is incremental methods, which can compute KPCs incrementally to handle online data processing. Chin and Suter proposed an incremental version of KPCA [14, 15], which is called IKPCA-RS for the notational simplicity. In IKPCA-RS, singular value decomposition is used to update an eigenfeature space incrementally for incoming data. However, IKPCA-RS may lead to high time complexity especially when dealing with high-dimensional data. In [16, 17], an incremental KPCA was presented based on the empirical kernel map. It is more efficient in memory requirement than the standard KPCA. However, it is only an approximate method and only suitable for polynomial kernel function. Inspired by the incremental PCA algorithm proposed by Hall et al. [18], Kimura et al. presented an incremental KPCA algorithm [19] in which an incremental updating algorithm for eigenaxes is derived based on a set of linearly independent data. Subsequently, some modified versions are proposed by Ozawa and Takeuchi et al. [20, 21]. Furthermore, in order to incrementally deal with data streams which are given in a chunk of multiple samples at one time, other extensions of KPCA were also successively presented [22–24]. Hallgren and Northrop [25] proposed an incremental KPCA (INKPCA) by applying rank one updates to the eigendecomposition of kernel matrix. However, INKPCA needs to store the whole data when evaluated on a new sample. Notably, incremental methods have the capacity of integrating new data, initially unavailable, in some way that maintains nonincreasing memory. However, to the best of our knowledge, most of these methods operate in the kernel space where all the samples are *implicitly* represented. This has two key limitations. *First*, a number of incremental methods may suffer from high computational cost. *Second*, the others can only capture the approximate KPCs rather than the accurate ones, which may affect the accuracy of its subsequent process.

Before continuing, a note on mathematical notations is given as follows. We use lower case and upper case letters (e.g., i, j, l, N) to denote scalars, lower case letters with the subscript (e.g., k_{ij}, α_i) to denote an element from a matrix or a vector, lower case bold letters (e.g., $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}$) to denote vectors, and upper case bold letters (e.g., $\boldsymbol{A}, \boldsymbol{C}, \boldsymbol{M}$) to denote matrices. We use $\boldsymbol{x}^T$ ($\boldsymbol{C}^{\mathrm{T}}$) to denote the transpose of a vector (matrix) and $\|\cdot\|$ to denote the L2-norm of a vector. Furthermore, we adopt $\{\boldsymbol{x_i}\}_{\mathrm{i=1}}^{\mathrm{N}}$ to denote a set, $[\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}]$ to denote a matrix with N column vectors and $[k_{ij}]_{1\le i\le M, 1\le j\le N}$ to denote a $M \times N$ matrix composed of the corresponding element k_{ij}. In this paper, $\boldsymbol{x_i}$ always denotes a column vector and the inner product between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is expressed as $\boldsymbol{x_i^T x_j}$. The lower case bold letter $\boldsymbol{\varphi}$ denotes a nonlinear mapping. The mapped sample $\boldsymbol{\varphi}(\boldsymbol{x})$ of the input sample $\boldsymbol{x}$ is a column vector.

To address these limitations, we propose a two-phase incremental KPCA (TP-IKPCA), where the mapped data is represented in an *explicit* form and KPCs are updated in an *explicit* space. The computational cost of the whole process is very low and the accuracy of KPCs can be theoretically guaranteed. An overview of TP-IKPCA is briefly illustrated in Figure 1. In this figure, $\{\boldsymbol{x_i}\}_{\mathrm{i=1}}^{\mathrm{N}}$ denotes the sample set in a d-dimensional input space and N denotes the total number of available samples. Let $\boldsymbol{\varphi}$ denote the nonlinear mapping which maps the sample set $\{\boldsymbol{x_i}\}_{i=1}^{N}$ into an h-dimensional implicit kernel space, resulting in the mapped sample set $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$. Here, h may be very large or even infinite, depending on the specific mapping. The TP-IKPCA includes two phases. *In the first phase*, we develop an incremental algorithm to capture standard orthogonal basis $\{\boldsymbol{\beta}_j\}_{j=1}^{r}$ of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x_i})\}_{i=1}^{N}$ and then *explicitly* obtain the projection vectors $\{\boldsymbol{y}_i\}_{i=1}^{N}$ of $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ by

$$\boldsymbol{y}_i = \left[\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r\right]^T \boldsymbol{\varphi}\left(\boldsymbol{x}_i\right), \tag{1}$$

where r denotes the number of a standard orthogonal basis $\{\boldsymbol{\beta}_j\}_{j=1}^{r}$. *In the second phase*, the existing incremental method

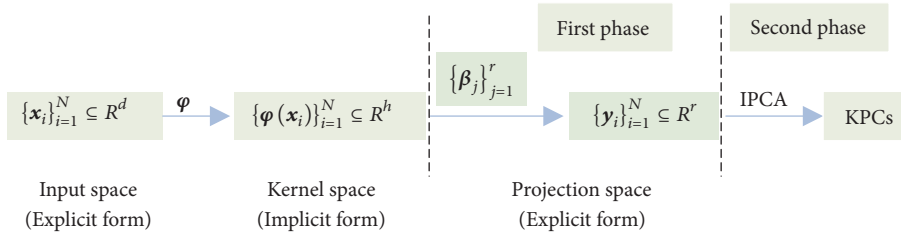


Figure 1: Overview of TP-IKPCA.

of PCA is employed to capture KPCs based on the explicit data $\{\boldsymbol{y}_i\}_{i=1}^{N}$ in the projection space. In the following sections, we will detail how to incrementally express the implicit mapped data $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ using an explicit form. We will also theoretically verify that performing PCA based on the implicitly mapped samples $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ is equivalent to that of based on the explicit projection vectors $\{\boldsymbol{y}_i\}_{i=1}^{N}$.

Here, we should clarify the relationship among some important quantities, including d, h, r (see Figure 1). In the case of KPCA or TP-IKPCA, the sample set $\{\boldsymbol{x}_i\}_{i=1}^{N}$ in a d-dimensional input space is firstly mapped into an h-dimensional kernel space by a nonlinear mapping $\boldsymbol{\varphi}$ and then a linear PCA is performed based on the mapped set $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$. Usually, h may be very large or even infinite, depending on the specific mapping $\boldsymbol{\varphi}$. So, we usually have $d \leq h$, which implies that the dimension of each mapped sample $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ is larger than its original dimension d. In the case of TP-IKPCA, $\{\boldsymbol{\beta}_j\}_{j=1}^{r}$ denote an orthonormal basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$, and $\{\boldsymbol{y}_i\}_{i=1}^{N}$ is the corresponding projection vectors of the mapped set $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ on the basis $\{\boldsymbol{\beta}_j\}_{j=1}^{r}$, which means that r is the dimension of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ and equals the dimension of each projected vector $\boldsymbol{y}_i$ $(i = 1, 2, \ldots, N)$. Generally, since the mapped data $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ have strong linear correlation in the h-dimensional kernel space, we have $r \leq h$, which means that a few components generally suffice to capture the nonlinear distribution of the data. Furthermore, if the dimension r of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ is high, r may be larger than the dimension d of the input space. However, if the mapped data $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ have strong linear correlation, which means r may be low and the dimension d of the input space is very high, we may get the contrast conclusion.

The main contributions of our work are fourfold: (1) Presenting an algorithm to express the mapped data in an explicit form. This will help for better understanding the distribution of the mapped data in the implicit kernel space. (2) Endowing KPCA with the capacity of handling dynamic dataset. (3) Compared to the standard KPCA, the computational complexity of TP-IKPCA is reduced from $O(N^3)$ to $O(r^3)$ and the storage complexity from $O(N^2)$ to $O(r^2)$, where N denotes the number of training samples and r is the number of bases of the subspace spanned by nonlinear mapped samples. Usually the assumption that $r \ll N$ is valid, which makes TP-IKPCA very convenient for processing large-scale datasets [26]. (4) In the testing stage, the feature extraction from one sample is faster than that of the batch KPCA, since TP-IKPCA only needs to calculate the kernel functions between the new sample and r selected training samples which forms the orthonormal basis.

The rest of the paper is organized as follows. Section 2 briefly introduces KPCA. In Section 3, we provide a theoretical analysis of the proposed TP-IKPCA method and elucidate the concrete steps for incrementally capturing KPCs based on the projection vectors in an explicit space. The effectiveness of TP-IKPCA is demonstrated in Section 4. Finally, the conclusions of our study are given in Section 5.

2. Kernel Principal Component Analysis (KPCA)

In this section, we briefly outline the standard procedure of KPCA. As mentioned above, in KPCA, the input sample set $\{\boldsymbol{x}_i\}_{i=1}^{N}$ is mapped into a kernel space by a nonlinear mapping $\boldsymbol{\varphi}$ and then a linear PCA is performed based on $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$ in the kernel space.

To obtain the eigenvectors in the kernel space, the covariance matrix is defined as

$$\boldsymbol{C} = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{\varphi}\left(\boldsymbol{x}_i\right) - \boldsymbol{c}\right)\left(\boldsymbol{\varphi}\left(\boldsymbol{x}_i\right) - \boldsymbol{c}\right)^T, \tag{2}$$

where $\boldsymbol{c} = (1/N)\sum_{i=1}^{N}\boldsymbol{\varphi}(\boldsymbol{x}_i)$. However, the eigendecomposition of $\boldsymbol{C}$ is hindered by the fact that the mapping function $\boldsymbol{\varphi}$ is implicit. To avoid the explicit calculation in the kernel space, a $N \times N$ kernel matrix $\boldsymbol{K}$ is defined, whose elements k_{ij} are determined by the virtue of the following kernel trick:

$$k_{ij} = \boldsymbol{\varphi}\left(\boldsymbol{x}_i\right)^T\boldsymbol{\varphi}\left(\boldsymbol{x}_i\right) = k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \quad i, j = 1, 2, \ldots, N. \tag{3}$$

where $k(\cdot, \cdot)$ is a kernel function that allows us to compute inner products in the kernel space [2].

Combining (2) and (3), Schölkopf et al. [2] derived the equivalent eigenvalue problem as follows:

$$\boldsymbol{K}\boldsymbol{\alpha} = N\lambda\boldsymbol{\alpha}, \tag{4}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$ denotes the column vector such that the orthogonal eigenvector $\boldsymbol{v}$ of the covariance matrix $\boldsymbol{C}$ satisfies

$$\boldsymbol{v} = \sum_{i=1}^{N}\alpha_i\boldsymbol{\varphi}\left(\boldsymbol{x}_i\right). \tag{5}$$

Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$ ($k \leq N$) denote the first k nonzero eigenvalues of $\boldsymbol{K}$ and $\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \ldots, \boldsymbol{\alpha}^k$ the corresponding complete set of eigenvectors (see (4)). We can obtain the corresponding eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ of $\boldsymbol{C}$ using (5).

Considering $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ need to be normalized, $\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \ldots, \boldsymbol{\alpha}^k$ need to satisfy

$$\lambda_l \left(\boldsymbol{\alpha}^l\right)^T \left(\boldsymbol{\alpha}^l\right) = 1 \quad l = 1, 2, \ldots, k. \tag{6}$$

For a test sample $\boldsymbol{x}$, the projection of $\boldsymbol{\varphi}(\boldsymbol{x})$ on the l-th nonlinear principal component can be computed by

$$\begin{aligned} d_l(\boldsymbol{x}) &= \left(\boldsymbol{v}_l\right)^T \boldsymbol{\varphi}(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i^l \left(\boldsymbol{\varphi}(\boldsymbol{x}_i)\right)^T \boldsymbol{\varphi}(\boldsymbol{x}) \\ &= \sum_{i=1}^{N} \alpha_i^l k(\boldsymbol{x}_i, \boldsymbol{x}). \end{aligned} \tag{7}$$

where α_i^l is the ith element of $\boldsymbol{\alpha}^l$; in other words, $\boldsymbol{\alpha}^l = [\alpha_1^l, \alpha_2^l, \ldots, \alpha_N^l]^{\boldsymbol{T}}$.

For the sake of simplicity, we assume that the mapped data $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ ($i = 1, 2, \ldots, N$) is zero-centered (see (2)). The detailed description of the centering processing is given in [2].

Of note, for KPCA, the kernel matrix $\boldsymbol{K}$ needs to be predefined before performing eigendecompositions. Since the size of $\boldsymbol{K}$ scales with N^2, a large memory space is required for a massive dataset. Additionally, the eigendecomposition of $\boldsymbol{K}$ involves a time complexity of $O(N^3)$. This can severely handicap the computation of KPCA on large datasets. In online processing applications, the arrival of a new sample requires adding a new row and a new column in $\boldsymbol{K}$, and eigendecomposition has to be constantly reevaluated for an ever-growing kernel matrix to update the kernel subspaces. Hence, the batch KPCA is not convenient for such applications.

3. Explicit Representation of the Mapped Data

At present, there have been many incremental algorithms for PCA [27–34]. However, it is difficult to directly extend them to KPCA because all mapped samples $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ are expressed implicitly in the kernel space. Obviously, once $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ can be expressed using an explicit form, it will be straightforward to extend incremental PCA to KPCA. In fact, the geometrical structure of $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ can be captured by using a standard orthogonal basis of the subspace spanned by all the samples $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ [26]. Hence, we aim to explicitly express $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ using an indirect way. This motivation comes from the following property shown in Theorem 1.

Let $\{\boldsymbol{\beta}_j\}_{j=1}^r$ denote an orthonormal basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$, and $\{\boldsymbol{y}_i\}_{i=1}^N$ is the corresponding projection vectors of $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ under $\{\boldsymbol{\beta}_j\}_{j=1}^r$ (see (1)). Theorem 1 is established as follows.

Theorem 1. *If linear PCA is performed based on $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ and $\{\boldsymbol{y}_i\}_{i=1}^N$, respectively, then their covariance matrices have the same nonzero eigenvalues, and those corresponding eigenvectors satisfy the following relationship:*

$$\boldsymbol{v}_l = \left[\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r\right] \boldsymbol{u}_l, \tag{8}$$

where $\boldsymbol{v}_l$ is the l-th eigenvector of the covariance matrix of $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ and $\boldsymbol{u}_l$ is the l-th eigenvector of the covariance matrix of $\{\boldsymbol{y}_i\}_{i=1}^N$. The proof is given in Appendix A.

Based on Theorem 1, linear PCA based on $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ can be converted into a linear PCA based on $\{\boldsymbol{y}_i\}_{i=1}^N$. So, if we can write $\{\boldsymbol{y}_i\}_{i=1}^N$ in an explicit form, then it will be easy to further extend KPCA using existing linear incremental algorithms. To incrementally obtain the orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^r$ and the projection vectors $\{\boldsymbol{y}_i\}_{i=1}^N$, we firstly introduce two correlative *lemmas*.

Lemma 2. *Let $\{\boldsymbol{\varphi}(\boldsymbol{x}_{bj})\}_{j=1}^r$ denote a basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$, then the orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^r$ can be determined using*

$$\left[\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r\right] = \left[\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})\right] \boldsymbol{D}, \tag{9}$$

where $\mathbf{D} = [\boldsymbol{\gamma}_1/\sqrt{\varepsilon_1}, \boldsymbol{\gamma}_2/\sqrt{\varepsilon_2}, \ldots, \boldsymbol{\gamma}_r/\sqrt{\varepsilon_r}]$. $\boldsymbol{\gamma}_j$ ($j = 1, 2, \ldots, r$) is the eigenvector of the kernel matrix $\boldsymbol{K}_{rr} = [\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})]^{\boldsymbol{T}} [\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})]$, scilicet $\boldsymbol{K}_{rr} = [k(\boldsymbol{x}_{bs}, \boldsymbol{x}_{bt})]_{1 \leq s,t \leq r}$. ε_j ($j = 1, 2, \ldots, r$) is the corresponding eigenvalue of $\boldsymbol{\gamma}_j$. The proof is given in Appendix B.

Based on Lemma 2, for any mapped sample $\boldsymbol{\varphi}(\boldsymbol{x}) \in \{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$, we can explicitly define its projection vector $\boldsymbol{y}$ under the orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^r$ as

$$\begin{aligned} \boldsymbol{y} &= \left[\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r\right]^T \boldsymbol{\varphi}(\boldsymbol{x}) \\ &= \boldsymbol{D}^T \left[\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})\right]^T \boldsymbol{\varphi}(\boldsymbol{x}) \\ &= \boldsymbol{D}^T \boldsymbol{k}_{bx}, \end{aligned} \tag{10}$$

where $\boldsymbol{k}_{bx} = [k(\boldsymbol{x}_{b1}, \boldsymbol{x}), k(\boldsymbol{x}_{b2}, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_{br}, \boldsymbol{x})]^T$. Obviously, using the kernel function $k(\cdot, \cdot)$ can complete the computation of $\boldsymbol{y}$.

However, the orthogonalization process using Lemma 2 is a batch-based method. Subsequently, when samples are added one by one, its computational cost is still very expensive. So, inspired by the Gram-Schmidt orthogonalization process [35], we designed an online algorithm for incrementally finding the orthonormal basis and the projection vectors.

Lemma 3. *Let $\{\boldsymbol{\varphi}(\boldsymbol{x}_{bj})\}_{j=1}^r$ denote a basis of the subspace spanned by $\{\varphi(\boldsymbol{x}_i)\}_{i=1}^N$ and $\{\boldsymbol{\beta}_j\}_{j=1}^r$ is the orthonormal basis obtained by (9). Suppose $\boldsymbol{x}_{N+1}$ denotes a new sample we have just included into our dataset. We derive the following properties.*

(1) If $\delta = k_{N+1} - \boldsymbol{k}_{b(N+1)}^T \boldsymbol{D}\boldsymbol{D}^T \boldsymbol{k}_{b(N+1)} = 0$, then $\{\boldsymbol{\beta}_j\}_{j=1}^r$ is the orthonormal basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N+1}$

and the projection vector $\boldsymbol{y}_{N+1}$ *of* $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$ *can be computed using (10). Here,* $k_{N+1} = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1})$ *and* $\boldsymbol{k}_{b(N+1)} = [k(\boldsymbol{x}_{b1}, \boldsymbol{x}_{N+1}), k(\boldsymbol{x}_{b2}, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_{br}, \boldsymbol{x}_{N+1})]^T$.

(2) If $\delta = k_{N+1} - \boldsymbol{k}_{b(N+1)}^T \boldsymbol{D}\boldsymbol{D}^T \boldsymbol{k}_{b(N+1)} \neq 0$, *then the orthonormal basis* $\{\boldsymbol{\beta}_j\}_{j=1}^{r+1}$ *of the subspace spanned by* $\{\boldsymbol{\varphi}(x_i)\}_{i=1}^{N+1}$ *can be obtained by*

$$\begin{aligned}&[\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r, \boldsymbol{\beta}_{r+1}] \\ &= [\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br}), \boldsymbol{\varphi}(\boldsymbol{x}_{b(r+1)})] \\ &\cdot \begin{bmatrix} \boldsymbol{D} & \dfrac{-\boldsymbol{D}\boldsymbol{D}^T \boldsymbol{k}_{b(N+1)}}{\sqrt{|\delta|}} \\ 0 & \dfrac{1}{\sqrt{|\delta|}} \end{bmatrix},\end{aligned} \tag{11}$$

where $\boldsymbol{x}_{b(r+1)} = \boldsymbol{x}_{N+1}$. *The projection vector* $\boldsymbol{y}_{N+1}$ *of* $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$ *can be computed by*

$$\boldsymbol{y}_{N+1} = \left[\boldsymbol{k}_{b(N+1)}^T \boldsymbol{D}, \ \sqrt{\delta}\right]^T. \tag{12}$$

Obviously, based on Lemma 3, it is straightforward to incrementally estimate the projection vector. Notably, the dimensionality of $\boldsymbol{y}_i$ $(i = 1, 2, \ldots, N)$ is smaller than that of $\boldsymbol{y}_{N+1}$ in the case of $\delta \neq 0$. In fact, based on the Gram-Schmidt orthogonalization process, let $\boldsymbol{y}_i'$ $(i = 1, 2, \ldots, N)$ denote the projection vector of $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ on the orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^{r+1}$, then $\boldsymbol{y}_i' = [\boldsymbol{y}_i^T, 0]^T$. The proof of Lemma 3 is provided in Appendix C.

Combining both Lemmas 2 and 3, we summarize the online algorithm, which incrementally finds the orthonormal basis and the projection vectors as follows.

Algorithm 4. An online algorithm for incrementally finding the orthonormal basis and the projection vectors.

Step 0 (initialization). For the time N=1, we found a sample $\boldsymbol{x}_1$. We suppose $k(\boldsymbol{x}_1, \boldsymbol{x}_1) \neq 0$. Let the set $\boldsymbol{S} = \{x_1\}$, $\boldsymbol{D} = 1/\sqrt{k(\boldsymbol{x}_1, \boldsymbol{x}_1)}$, and $\boldsymbol{y}_1 = \sqrt{k(\boldsymbol{x}_1, \boldsymbol{x}_1)}$.

Step 1. Calculate δ for a new sample $\boldsymbol{x}_{N+1}$ according to

$$\delta = k_{N+1} - \boldsymbol{k}_{b(N+1)}^T \boldsymbol{D}\boldsymbol{D}^T \boldsymbol{k}_{b(N+1)}, \tag{13}$$

where $k_{N+1}, \boldsymbol{D}$ and $\boldsymbol{k}_{b(N+1)}$ have the same definition as in Lemma 3.

Step 2. If $\delta = 0$, then $\boldsymbol{y}_{N+1} = \boldsymbol{D}^T \boldsymbol{k}_{b(N+1)}$ and return to Step 1.

Step 3. If $\delta \neq 0$, then $\boldsymbol{S} = \boldsymbol{S} \cup \{\boldsymbol{x}_{N+1}\}$ and update $\boldsymbol{y}_{N+1}$ using (12). Finally, update $\boldsymbol{D}$ using (14) and return to Step 1.

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D} & \dfrac{-\boldsymbol{D}\boldsymbol{D}^T \boldsymbol{k}_{b(N+1)}}{\sqrt{|\delta|}} \\ 0 & \dfrac{1}{\sqrt{|\delta|}} \end{bmatrix}. \tag{14}$$

Obviously, if we map all the samples of $\boldsymbol{S}$ into the kernel space and get the dataset $\{\boldsymbol{\varphi}(\boldsymbol{x}_{bj}) \mid \boldsymbol{x}_{bj} \in \boldsymbol{S}\}$, then the mapped samples $\{\boldsymbol{\varphi}(\boldsymbol{x}_{bj}) \mid \boldsymbol{x}_{bj} \in \boldsymbol{S}\}$ are linearly independent. Furthermore, we can get an orthonormal basis based on $\{\boldsymbol{\varphi}(\boldsymbol{x}_{bj}) \mid \boldsymbol{x}_{bj} \in \boldsymbol{S}\}$ and $\boldsymbol{D}$ (see (9) or (11)). In fact, taking into account the actual calculation error, we usually use a very small threshold value θ to decide performing Step 2 or Step 3 in Algorithm 4. In other words, if $\delta < \theta$, we perform step 2; otherwise, we perform Step 3.

4. Incremental Learning of KPCA

In this section, we will outline the incremental learning method of KPCA based on the incremental version of PCA (IPCA) proposed by Hall et al. [18]. The key difference between our method and Hall et al.'s method is that the dimensionality of the projection vector $\boldsymbol{y}_i$ $(i = 1, 2, \ldots, N)$ in our case is not constant; hence we further adapt IPCA to our aim and address this limitation.

4.1. Description of IKPCA. Given a sample set $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ and its corresponding projection vector set $\{\boldsymbol{y}_i\}_{i=1}^N$ (see the Section 3), we assume we have already built a set of eigenvectors $\{\boldsymbol{u}_l\}_{l=1}^p$ and its corresponding matrix $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_p]$ with the $\{\boldsymbol{y}_i\}_{i=1}^N$ set as an input. Note that we have $p \leq r$ where r denotes the dimension of $\boldsymbol{y}_i$ $(i = 1, 2, \ldots, N)$. Let $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p)$ denote the corresponding matrix of eigenvalues and $\overline{\boldsymbol{y}}$ the mean vector. Incremental building requires updating these eigenvectors when a new input sample $\boldsymbol{y}_{N+1}$ is obtained, which is the projection vector of $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$. Obviously, the dimensionality of $\boldsymbol{y}_{N+1}$ may be larger than that of $\boldsymbol{y}_i$ $(i = 1, 2, \ldots, N)$ (see (12)). When their dimensionalities are identical, we denote $r_{N+1} = r$, otherwise, $r_{N+1} \neq r$. Firstly, we update the mean:

$$\overline{\boldsymbol{y}} = \begin{cases} \dfrac{1}{N+1}\left(N\overline{\boldsymbol{y}} + \boldsymbol{y}_{N+1}\right) & r_{N+1} = r \\ \dfrac{1}{N+1}\left(N\begin{pmatrix}\overline{\boldsymbol{y}} \\ \boldsymbol{0}\end{pmatrix} + \boldsymbol{y}_{N+1}\right) & r_{N+1} \neq r, \end{cases} \tag{15}$$

where $(\overline{\boldsymbol{y}}, 0)^T$ means adding one zero to the original vector $\overline{\boldsymbol{y}}$. Then we update the set of eigenvectors $\{\boldsymbol{u}_l\}_{l=1}^p$ by adding a new vector $\boldsymbol{y}_{N+1}$ and applying a rotational transformation. In order to do this, we first compute the orthogonal residual vector:

$$\boldsymbol{h}_{N+1} = \begin{cases} (\boldsymbol{U}\boldsymbol{\eta}_{N+1} + \overline{\boldsymbol{y}}) - \boldsymbol{y}_{N+1} & r_{N+1} = r \\ \left(\begin{bmatrix}\boldsymbol{U} \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{\eta}_{N+1} + \overline{\boldsymbol{y}}\right) - \boldsymbol{y}_{N+1} & r_{N+1} \neq r, \end{cases} \tag{16}$$

where $\boldsymbol{\eta}_{N+1}$ is computed by

$$\boldsymbol{\eta}_{N+1} = \begin{cases} \boldsymbol{U}^T\left(\boldsymbol{y}_{\boldsymbol{N+1}} - \overline{\boldsymbol{y}}\right) & r_{N+1} = r \\ \begin{bmatrix}\boldsymbol{U} \\ 0\end{bmatrix}^T \left(\boldsymbol{y}_{N+1} - \overline{\boldsymbol{y}}\right) & r_{N+1} \neq r. \end{cases} \tag{17}$$

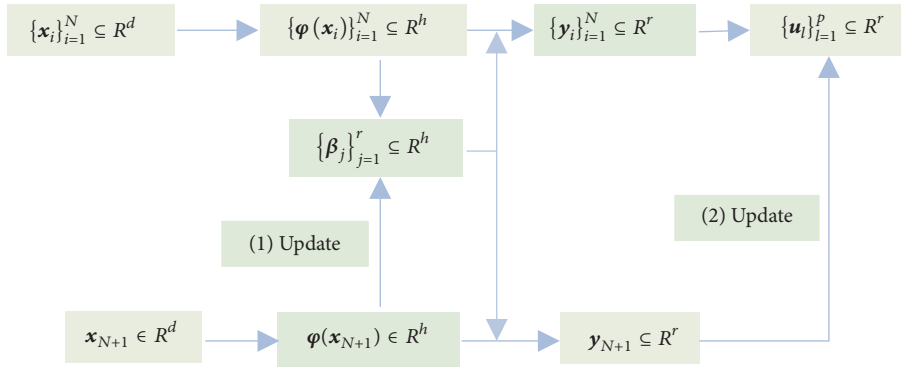


Figure 2: Flowchart of TP-IKPCA.

Subsequently, we normalize $\boldsymbol{h}_{N+1}$ to obtain $\widehat{\boldsymbol{h}}_{N+1} = \boldsymbol{h}_{N+1}/\|\boldsymbol{h}_{N+1}\|$ for $\|\boldsymbol{h}_{N+1}\| > 0$ and $\widehat{\boldsymbol{h}}_{N+1} = 0$ otherwise. The new matrix of eigenvectors $\boldsymbol{U}'$ is computed by

$$\boldsymbol{U}' = \begin{cases} \left[\boldsymbol{U}, \widehat{\boldsymbol{h}}_{N+1}\right]\boldsymbol{T} & r_{N+1} = r_N \\ \left[\begin{bmatrix}\boldsymbol{U} \\ 0\end{bmatrix}, \widehat{\boldsymbol{h}}_{N+1}\right]\boldsymbol{T} & r_{N+1} \neq r_N, \end{cases} \tag{18}$$

where $\boldsymbol{T} \in R^{(p+1)\times(p+1)}$ is a rotation matrix with dimension p+1. $\boldsymbol{T}$ is the solution of the eigenproblem of the following form [18, 36]:

$$\boldsymbol{H}\boldsymbol{T} = \boldsymbol{T}\boldsymbol{\Lambda}'. \tag{19}$$

We compose $\boldsymbol{H} \in \mathrm{R}^{(p+1)\times(p+1)}$ as

$$\boldsymbol{H} = \frac{N}{N+1}\begin{bmatrix}\boldsymbol{\Lambda} & 0 \\ 0^T & 0\end{bmatrix} + \frac{N}{(N+1)^2}\begin{bmatrix}\boldsymbol{\eta}\boldsymbol{\eta}^T & \boldsymbol{\tau}\boldsymbol{\eta} \\ \boldsymbol{\tau}\boldsymbol{\eta}^T & \boldsymbol{\tau}^2\end{bmatrix}. \tag{20}$$

where $\boldsymbol{\tau} = \widehat{\boldsymbol{h}}_{N+1}^T(\boldsymbol{y}_{N+1} - \overline{\boldsymbol{y}})$ and $\boldsymbol{\eta} = \boldsymbol{\eta}_{N+1}$.

Broadly, the procedure of our incremental method is similar to IPCA presented by Hall et al. [18]. Only under the condition $r_{N+1} \neq r_N$, we add one zero (zero vector) to the corresponding variant (matrix).

Once we determined the principal direction set $\{\boldsymbol{u}_l\}_{l=1}^p$, for a test sample $\boldsymbol{x}$, the projection of $\boldsymbol{\varphi}(\boldsymbol{x})$ onto the l-th nonlinear principal direction $\boldsymbol{v}_l$ can be obtained using the formulas in (8) and (10):

$$\begin{aligned} d_l(\boldsymbol{x}) &= (\boldsymbol{v}_l)^T\boldsymbol{\varphi}(\boldsymbol{x}) = (\boldsymbol{u}_l)^T(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r)^T\boldsymbol{\varphi}(\boldsymbol{x}) \\ &= (\boldsymbol{u}_l)^T\boldsymbol{D}^T\boldsymbol{k}_{bx} = (\boldsymbol{u}_l)^T\boldsymbol{y}. \end{aligned} \tag{21}$$

4.2. Framework of TP-IKPCA. Based on the analysis in Section 3 and in Section 4.1, we present the flowchart of TP-IKPCA in Figure 2. Here, $\{\boldsymbol{x}_i\}_{i=1}^N$ denotes the input sample set and $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ represents its corresponding mapped set by an implicit nonlinear mapping $\boldsymbol{\varphi}$. Firstly, an orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^r$ of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ (see (9) or (11)) and the corresponding projection vectors set $\{\boldsymbol{y}_i\}_{i=1}^N$ (see (10)) are obtained. Subsequently, a set of eigenvectors $\{\boldsymbol{u}_l\}_{l=1}^p$ of $\{\boldsymbol{y}_i\}_{i=1}^N$ are built. Then, for a new coming sample $\boldsymbol{x}_{N+1}$, its mapped sample $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$ is used to update the orthonormal basis $\{\boldsymbol{\beta}_j\}_{j=1}^r$ and its projection vector $\boldsymbol{y}_{N+1}$ is computed using Algorithm 4. Finally, based on the IKPCA described in Section 4.1, the eigenvector set $\{\boldsymbol{u}_l\}_{l=1}^p$ is updated using $\boldsymbol{y}_{N+1}$. It can be seen from Figure 2 that TP-IKPCA includes two main steps. The first step is based on incremental learning algorithm that represents the mapped data using an explicit form (see Algorithm 4). The second step incrementally computes the principal components of $\{\boldsymbol{y}_i\}_{i=1}^{N+1}$ using IKPCA (see Section 4.1).

The dimensions d,h, and r of the input, kernel, and projection spaces, respectively, generally satisfy the following inequalities: $d \leq h$ and $r \leq h$ (see Section 1). Here, we need to focus on the meaning of p. In this paper, p denotes the number of the eigenvectors derived from the covariance matrix of the projection vector set $\{\boldsymbol{y}_i\}_{i=1}^N$. In other words, p is the number of the principal components of $\{\boldsymbol{y}_i\}_{i=1}^{N+1}$. We note that the number of the principal components, that is p, should be also smaller than the dimension r of the projection space. In summary, we have $p \leq r \leq h$.

4.3. Complexity Analysis of TP-IKPCA. Suppose that given the current sample set $\{\boldsymbol{x}_i\}_{i=1}^N$, the dimension of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ is r. When a new sample $\boldsymbol{x}_{N+1}$ arrives, TP-IKPCA first conducts Algorithm 4 where the most time-consuming step is to compute $\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)}$ appearing in (11)-(14). The time complexity for this step is $O(r^2)$ since $\boldsymbol{D}$ is an $r \times r$ orthogonal transformation matrix and $\boldsymbol{k}_{b(N+1)}$ is an N-dimensional vector. Then, TP-IKPCA incrementally computes the principal components using IKPCA (see Section 4.1). In this step, the computation of the eigendecomposition of matrix $\boldsymbol{H} \in R^{(p+1)\times(p+1)}$ in (20) is most time-consuming. We can define the upper bound for its time complexity as $O(p^3) \leq O(r^3)$ since we have $p \leq r$. Considering the above two steps, the overall time complexity of TP-IKPCA in worst case can be estimated as $O(r^3)$. In fact, $r \ll N$ is usually tenable [26, 37], which makes TP-IKPCA convenient for processing large-scale or online datasets. Meanwhile, TP-IKPCA requires storing an

$r \times r$ orthogonal transformation matrix $\boldsymbol{D}$ (see (9)), which only involves a space complexity of $O(r^2)$. Especially in the testing stage, obtaining the principal component of a new sample only needs to calculate the kernel functions between the new sample and the r old training samples that compose the orthonormal basis (see (21)), which results in improving the computing speed.

5. Experiments

We evaluated and compared the performance of TP-IKPCA on synthetic and real datasets with several typical KPCA-based approaches in terms of accuracy and time complexity. The comparison methods include (1) conventional batch mode KPCA, (2) incremental KPCA [14, 15] with reduced-set (IKPCA-RS), and (3) the recently developed incremental KPCA [25] based on rank one updates to the eigendecomposition of kernel matrix (INKPCA). The time complexity can be captured by two aspects: (1) time required for learning the training data; (2) r, i.e., the number of orthonormal basis elements of the subspace spanned by the mapping training data. Usually, a smaller r indicates a reduced time complexity of TP-IKPCA (see Section 4.3). At the same time, we also used two different measures to evaluate the accuracy of TP-IKPCA. The first one is the correlation coefficient between two corresponding principal components (PCs) of TP-IKPCA and KPCA. Since KPCA is performed using all training data in a batch learning model, the PCs of KPCA are the target that TP-IKPCA needs to capture. Ideally, the PCs of TP-IKPCA should be identical with those of KPCA. Therefore, the correlation coefficient between two corresponding PCs is evaluated to show how accurate is TP-IKPCA in comparison to batch KPCA. Specifically, let $\boldsymbol{v}_l$ denote the l-th PC of KPCA after learning all of the N samples and $\boldsymbol{v}_l^{\boldsymbol{TP}}(i)$ is the l-th PC of TP-IKPCA after learning i $(i \leq N)$ samples, we define the correlation coefficient (corr) by

$$corr\left(\boldsymbol{v}_l, \boldsymbol{v}_l^{\boldsymbol{TP}}(\boldsymbol{i})\right) = \frac{\left(\boldsymbol{v}_l\right)^T \boldsymbol{v}_l^{\boldsymbol{TP}}(\boldsymbol{i})}{\left\|\boldsymbol{v}_l\right\| \left\|\boldsymbol{v}_l^{\boldsymbol{TP}}(\boldsymbol{i})\right\|}, \tag{22}$$

The specific computation in (22) can be deduced from (5) and (8). The second measure is to compare the effectiveness of TP-IKPCA and KPCA. Here, for two-dimensional synthesized datasets, we adopt the contour lines of PCs as an evaluation measure. For real datasets, we adopt the practical denoising effect.

5.1. Synthesized Data. In this experiment, we use two-dimensional nonlinear synthetized data to evaluate the accuracy and memory space efficiency of KPCA, IKPCA-RS, INKPCA, and our proposed TP-IKPCA. The data is generated by:

$$y = \left(\frac{x^2}{2} + 1\right) + 0.2\xi, \quad \xi \sim N(0,1), \; x \sim U[0,1], \tag{23}$$

where $N(0,1)$ denotes the standard Gaussian distribution and $U[0,1]$ is the uniform distribution in $[0,1]$. In this experiment, the kernel function is the polynomial kernel form, namely, $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i^T x_j})^{\boldsymbol{m}}$. Here, the parameter m=2.

Table 1: Learning time (sec) for proposed and comparison methods.

	Training stage	Testing stage
KPCA	1.067 ± 0.03	0.203 ± 0.006
IKPCA-RS	0.201 ± 0.0032	0.007 ± 0.0003
INKPCA	0.145 ± 0.0041	0.201 ± 0.004
TP-IKPCA	**0.087 ± 0.0026**	**0.004 ± 0.0002**
Notice	The training number N=500 and the basis number r=9 in TP-IKPCA.	

The contour lines of the first three PCs obtained by each method for N=500 training samples are illustrated in Figure 3 (Top to bottom: KPCA, TP-IKPCA, IKPCA-RS, and INKPCA), where red dots represent samples and green lines represent the contour lines of the corresponding PCs. λ_i $(i = 1, 2, 3)$ denotes the eigenvalue of the corresponding PC. Figure 3 shows no visually discernible differences between our method and the 3 comparison methods. In other words, their contour lines are very similar. Furthermore, the differences in eigenvalues λ_i are also very small. Therefore, it can be concluded that all of the results derived from the three incremental algorithms: IKPCA-RS, INKPCA, and our proposed TP-IKPCA, follow the ground truth results closely.

Figure 4 shows the evolution curves of the correlation coefficients between the first three PCs obtained by TP-IKPCA, IKPCA-RS, INKPCA, and their corresponding PCs obtained by KPCA when increasing the number of training samples. Figure 4 shows for different incremental algorithms that the resulting correlation coefficient gradually converges to 1 as the number of training samples increases. It indicates that the PCs obtained by TP-IKPCA, IKPCA-RS, and INKPCA gradually approximate those obtained by batch mode KPCA with high accuracy.

Table 1 shows the average training time for learning from 500 training samples and testing time for extracting features from 100 testing samples (in seconds). We repeated this procedure 20 times and reported the averaged training and testing times as well as the corresponding standard deviations.

From Table 1, we can clearly see that when using the synthesized data, all of the three incremental KPCA algorithms have faster training speed than KPCA due to low-dimensional features as well as simple distribution of this data. Among these incremental variants, our proposed TP-IKPCA leads to fastest training speed. From the perspective of testing speed, our method can perform prediction in the least time. This can be explained by the fact that the learning time of TP-IKPCA depends on the size of orthonormal basis r (=9) while KPCA and INKPCA both depend on the total number of training samples (=500 in our case). IKPCA-RS also leads to fast testing speed since it uses a reduced set. However, it performs slower than our method in training speed since seeking a set of approximate preimages when new samples arrive using certain optimization techniques or fixed-point iteration is time-consuming.

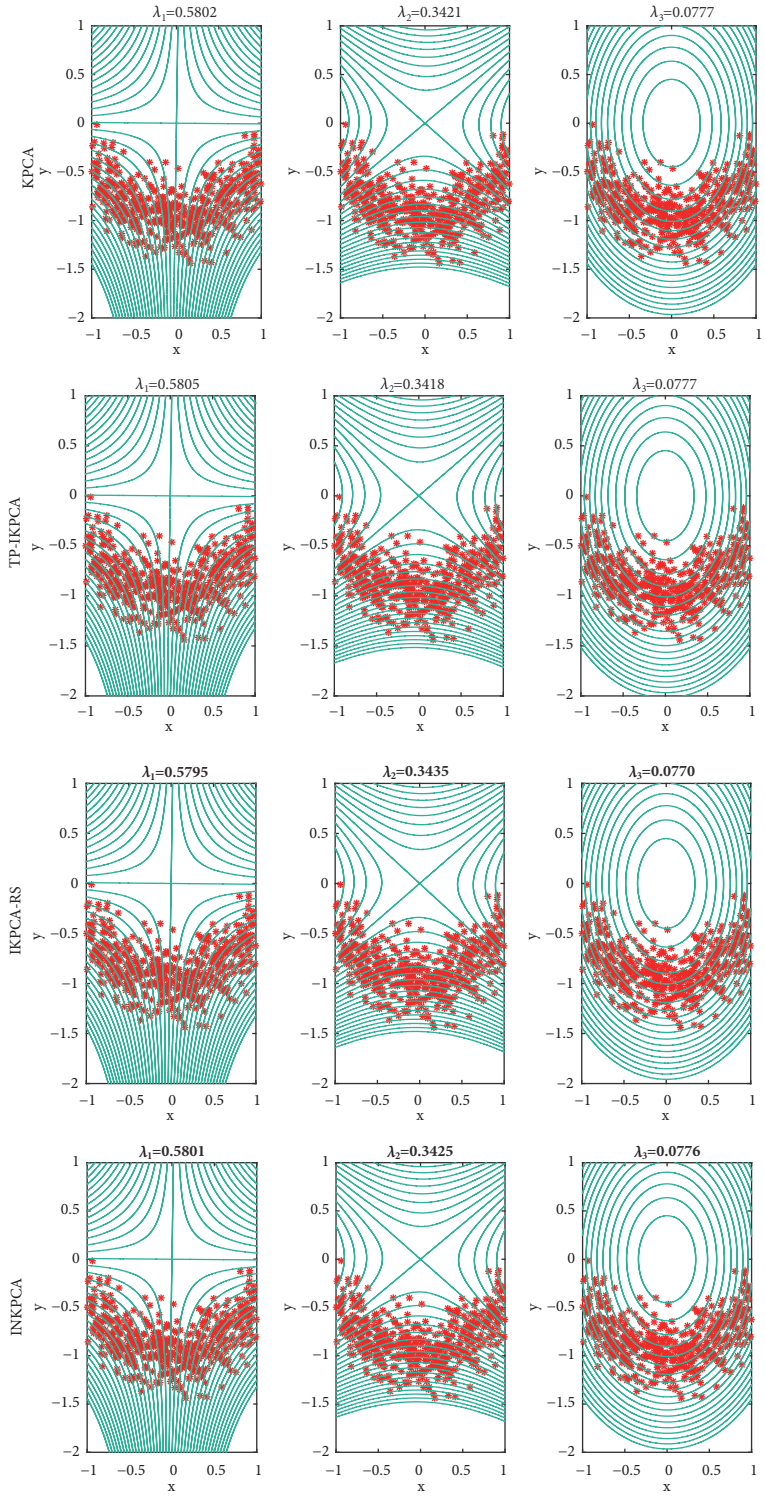
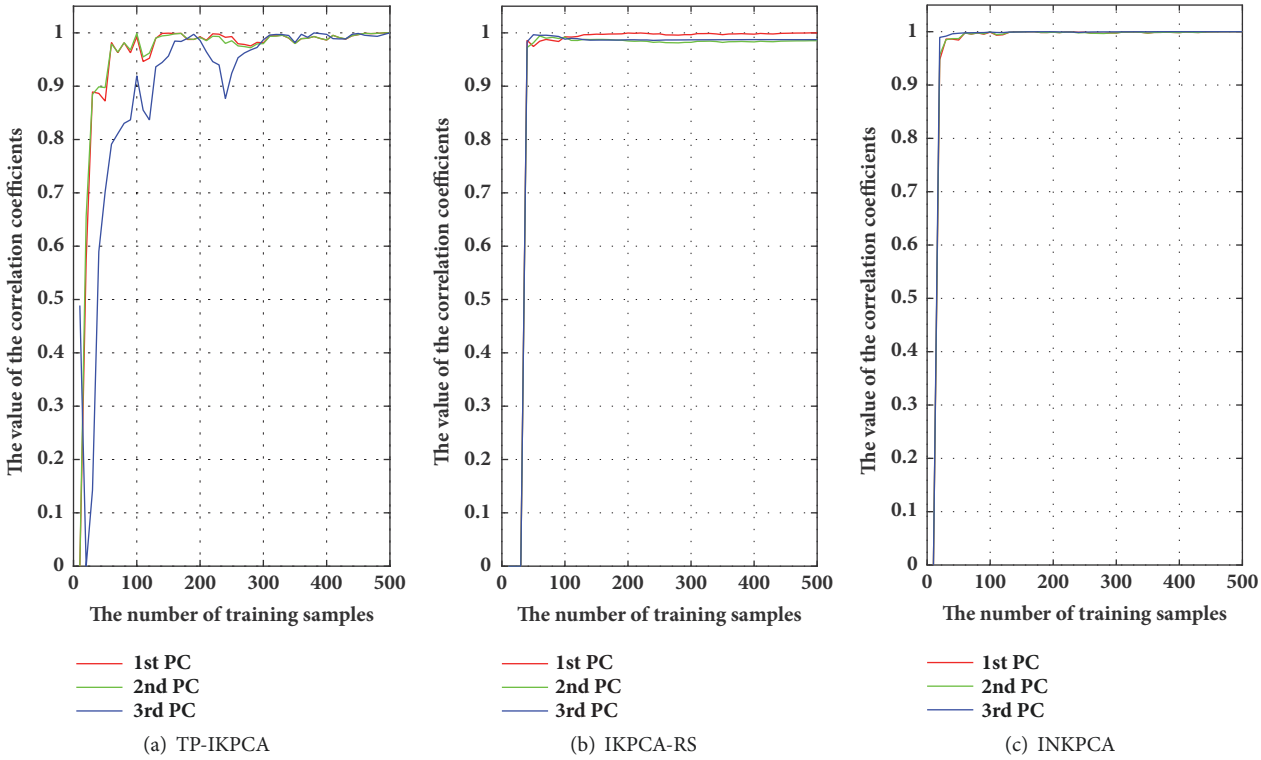


Figure 3: Synthesized data including 500 samples and the contours of the first three principal components drawn using a polynomial kernel. The first row is from KPCA, the second row is from TP-IKPCA, the third row is from IKPCA-RS, and the forth row is from INKPCA. Data points are represented by red dots "∗" and the green lines are the contour lines of constant value of the first three principal components.



(a) TP-IKPCA (b) IKPCA-RS (c) INKPCA

Figure 4: Evolution of the correlation coefficients between the first three PCs of three incremental algorithms and KPCA when increasing the number of training samples.

In what follows, we design two experiments to investigate the behavior of our algorithm when the number of training samples increases. Firstly, in Figure 5, we plot the variation curve of the number of basis, i.e., r, against the number of training samples (N). From this result, we notice that in the beginning, r gradually increases with N. However, after $N \geq 21$, r stops increasing. This shows that the mapped data have strong linear correlation in the kernel space. Hence, although the number of training samples continues to increase, the number of basis remains stable. More importantly, the computational complexity of TP-IKPCA becomes a constant $O(9^3)$ when $N \geq 21$, which is a significant improvement over standard KPCA (in the order of N^3) —particularly when the number of training samples becomes very large.

Then, we compute the acceleration ratio which represents the ratio of the time consumed by KPCA to extract features from 100 test samples to the time consumed by TP-IKPCA. The resulting variation of acceleration ratio for testing speed with respect to the number of training samples is shown in Figure 6. Obviously, a larger ratio indicates a faster test speed of TP-IKPCA compared with KPCA. Figure 6 also shows that the larger the number of training samples, the larger the ratio, implying that TP-IKPCA can significantly improve test speed compared with KPCA.

5.2. MNIST Data. In this section, we consider an image processing application where we process the MNIST database of handwritten digits (http://yann.lecun.com/exdb/mnist/). The database consists of handwritten digits from 0 to 9. Each

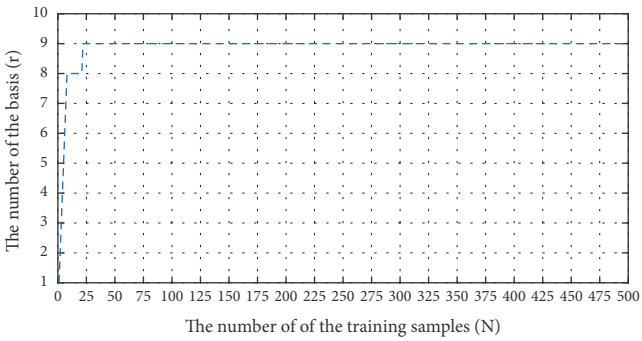


Figure 5: Variation of the number of basis (r) with respect to the number of training samples (N).

digit set includes training set and testing set. Each digit is represented by a 28-by-28 image. In order to evaluate the performance of different approaches, we carried out image denoising experiment to the even number. Firstly, for each digit, we randomly select 500 training samples and 100 testing samples and then add the Gaussian noise and the salt-and-pepper noise, respectively, to the testing images. The mean and variance for the Gaussian noise are 0 and 0.2, respectively, while the level of the salt-and-pepper noise is 0.3. Next, we estimate the first sixteen principal components using KPCA, IKPCA-RS, INKPCA, and our proposed TP-IKPCA, respectively. Finally, we perform denoising experiments on all corrupted testing samples and reconstruct

Figure 6: Changes of the ratio of the test speed with respect to the training sample numbers (N).

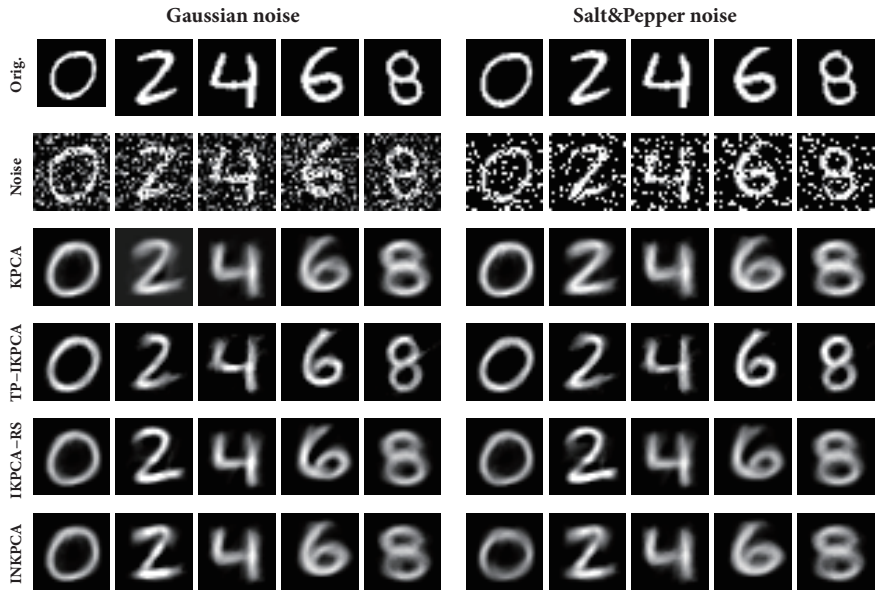


Figure 7: Restoration results by TP-IKPCA, IKPCA-RS, INKPCA, and KPCA.

them using the reconstructive scheme presented in [38]. In the experiment, we use the Gaussian function $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2/\sigma^2)$ as the kernel function in each method and set the bandwidth $\sigma = 0.2$.

Figure 7 shows some restoration results from the corrupted images by conducting KPCA, TP-IKPCA, IKPCA-RS and INKPCA, respectively. It can be seen from these figures that all of these methods can eliminate noise and reconstruct the images well. Therefore, we can draw the conclusion that different incremental KPCA algorithms can closely approximate batch mode KPCA in reconstruction performance with high accuracy.

Figure 8 shows the evolution curves of the correlation coefficients between the first three PCs by TP-IKPCA, IKPCA-RS, INKPCA, and their corresponding ones by KPCA when the number of training samples increases, where the horizontal axis denotes the number of training samples and the vertical axis is the correlation coefficient computed by (22). Figure 8 shows that all incremental KPCA algorithms lead to good approximation accuracies for the first three PCs since the correlation coefficients gradually converge to 1 as the number of training samples increases. The results indicate that incremental KPCA algorithms can gradually capture PCs in real high-dimensional datasets with good approximation accuracy.

We also display in Table 2 the average training and testing times (in seconds) for training 500 samples and extracting features from 100 testing samples, respectively. We repeated this process 20 times and reported the average values and the corresponding standard deviations.

Firstly, we analyze the training results shown in Table 2. We can derive the following observations: (1) IKPCA-RS consumes much more time in training than other approaches, including batch model KPCA. These results differ from those obtained when using the synthesized data where IKPCA-RS ran faster than batch mode KPCA. Through inspecting the experiments, we found that IKPCA-RS iterates reduced-set expansions many times when computing the preimages for compression. As a result, when handling data with a large number of features (=784 in our case), the calculation of

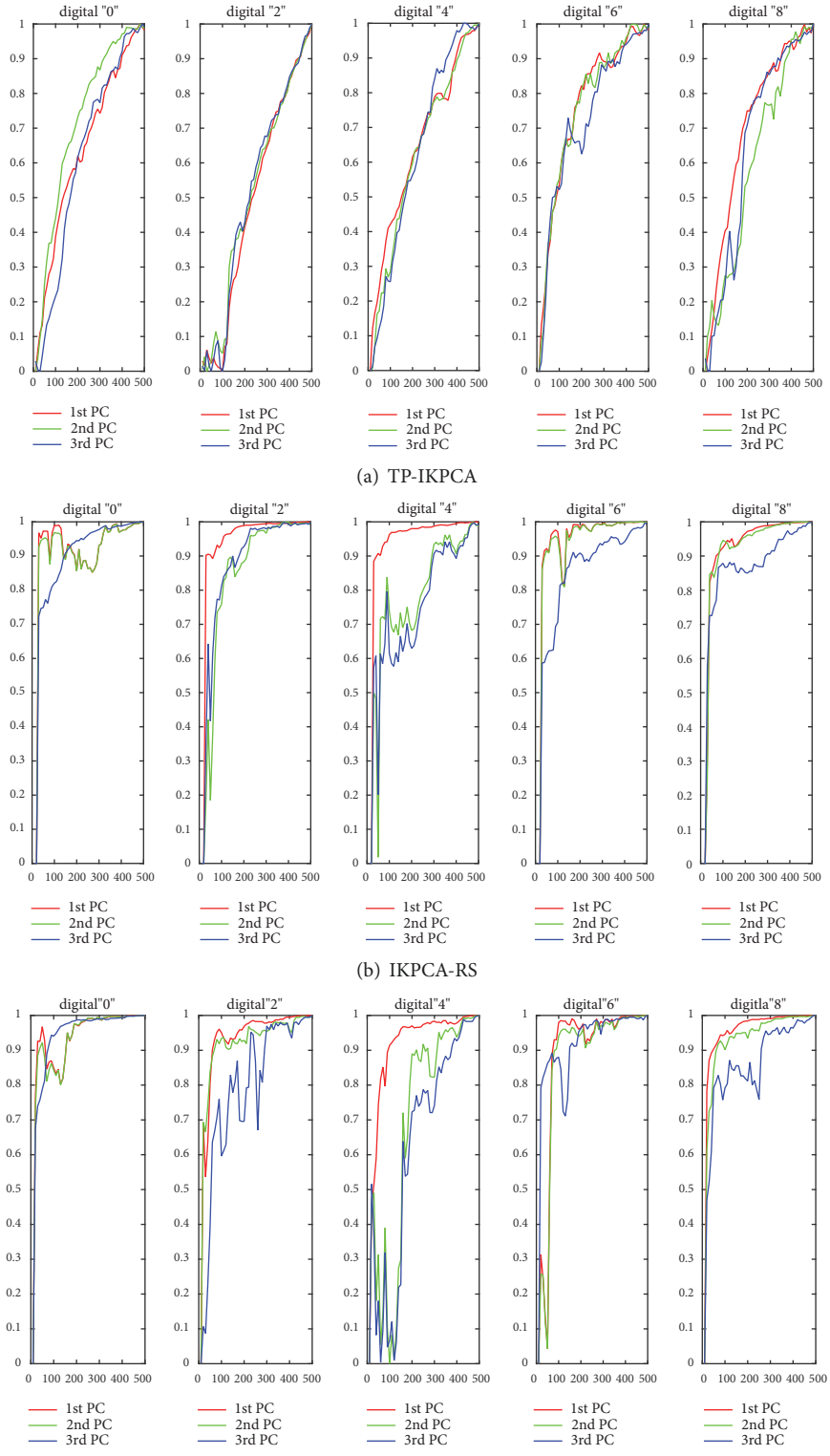


FIGURE 8: Evolution curves of the correlation coefficients between the first three PCs of (a) TP-IKPCA, (b) IKPCA-RS, (c) INKPCA, and KPCA as the number of training samples increases (x-axis).

Table 2: Experimental results of learning time on different digit (sec).

	Training stage					Testing stage				
	"0"	"2"	"4"	"6"	"8"	"0"	"2"	"4"	"6"	"8"
KPCA	1.75 ± 0.055	1.74 ± 0.018	1.74 ± 0.021	1.76 ± 0.036	1.74 ± 0.030	0.358 ± 0.022	0.351 ± 0.013	0.357 ± 0.012	0.352 ± 0.011	0.354 ± 0.011
IKPCA-RS	43.68 ± 0.049	41.32 ± 0.038	43.21 ± 0.068	44.75 ± 0.074	42.87 ± 0.059	0.221 ± 0.009	**0.243** ± 0.011	0.208 ± 0.014	0.169 ± 0.017	0.214 ± 0.008
INKPCA	23.67 ± 0.032	22.13 ± 0.038	23.91 ± 0.041	23.10 ± 0.047	22.13 ± 0.035	0.360 ± 0.023	0.358 ± 0.014	0.353 ± 0.012	0.355 ± 0.012	0.352 ± 0.011
TP-IKPCA	**1.89** ± 0.019	**3.06** ± 0.017	**1.67** ± 0.043	**1.40** ± 0.024	**2.27** ± 0.031	**0.181** ± 0.012	0.251 ± 0.008	**0.162** ± 0.016	**0.144** ± 0.013	**0.199** ± 0.010
r	242	351	219	190	282	242	351	219	190	282

preimages increases the computational load. This observation is in line with the conclusion drawn in [15]. (2) INKPCA needs to apply rank one update to iteratively calculate eigenvalues and eigenvectors associated with kernel matrix when new data arrive. Although INKPCA consumes more time than batch mode KPCA, it does not need to store the whole kernel matrix in memory and has the advantage of better handling massive data where KPCA rapidly becomes infeasible. (3) As for our proposed TP-IKPCA, it runs much faster than other incremental algorithms, including IKPCA-RS and INKPCA, excluding a few cases where our algorithm may be slower than batch mode KPCA. This can be explained by the computational complexity of TP-IKPCA of the order of r^3. In this experiment, the linear correlation between the mapped digital images is very weak, which results in a very large r and even approximates the number of training samples.

From the testing results shown in Table 2, we can conclude the following: (1) The testing speeds of KPCA and INKPCA are similar since INKPCA cannot select a few yet important samples from the whole data but still makes use of all available samples when calculating the projections of new data. Therefore, their speed is proportional to the total number of the training samples. (2) Regardless of specific digit, the testing speed of IKPCA-RS and TP-IKPCA are much faster than that of KPCA and INKPCA since both methods are able to reduce the number of samples used for kernel evaluation although they adopt different strategies. The testing time of TP-IKPCA is proportional to the size of basis r. In this experiment, the size of basis r is smaller than the number of training samples, thus leading to an improvement in the test speed compared with KPCA. Considering the training time, our proposed TP-IKPCA is obviously preferred over IKPCA-RS.

In what follows, we gradually increase the number of training samples and summarize the training and testing time required by TP-IKPCA and standard KPCA. We find from extensive experiments that the computational superiority of TP-IKPCA over KPCA increases with the number of training samples. Taking the experiments on digit "0" as an example, we repeated this evaluation 20 times and recorded the resulting training and testing time (in seconds) required by TP-IKPCA and KPCA under different training sample size N. Finally, the averaged time and the standard deviation are shown in Table 3 where the training ratio in Table 3 denotes the ratio of training time of KPCA to that of TP-IKPCA, given a total number of N samples. In a similar way, the testing ratio represents the ratio of testing time of KPCA to that of TP-IKPCA when extracting features from 100 test samples.

Based on Table 3, we can make the following observations: (1) As the number of training samples continues to increase, the number of basis r tends to increase as well. However, the increasing speed of r gradually decreases, which indicates that the larger the size of the training set, the stronger the correlation between the samples. (2) With the increasing of the training set size N, the training time of both KPCA and TP-IKPCA also increases gradually. However, the increasing speed of TP-IKPCA is much slower than that of KPCA. The reason lies in that the time complexity of TP-IKPCA has a close relationship with the number of basis r while that of KPCA depends on the total number of training samples N. As N *increases*, the increasing speed of r progressively decreases since most of the correlation structure among data has been revealed. We also derive a similar conclusion from the ratio's evolution in the training stage with respect to the changes of the number of training samples, where the ratio gradually increases with N in the training stage. (3) As N increases, the testing time of KPCA and TP-IKPCA both increase gradually. However, the increasing speed of TP-IKPCA is much slower than that of KPCA, which is also reflected by the ratio's evolution in the testing stage. The reason is that the test speed of TP-IKPCA is closely related to the number of basis r and the increasing speed of r gradually becomes slower with the increasing of N. Based on the above analysis, we conclude that TP-IKPCA does significantly improve the computational complexity of KPCA. Moreover, TP-IKPCA can deal with dynamic dataset due to its "incremental" nature.

6. Conclusion

In this paper, we proposed a novel incremental feature extraction method termed as TP-IKPCA which endowed KPCA with the capability of handling dynamic or large-scale datasets. The proposed TP-IKPCA differs from the existing incremental approaches in providing an explicit form of the mapped data and the updating process of KPCs is also performed in an explicit space. Specifically, TP-IKPCA is implemented in two phases. First, an incremental algorithm is given to explicitly project the mapped samples in the kernel space. Second, we employed the existing incremental method of PCA to capture KPCs based on the explicit data in the projection space. The computational complexity of TP-IKPCA has a close relationship with the size of basis r of the subspace spanned by the mapped training samples. Usually, r is much smaller than the number of training samples N, and thus TP-IKPCA can greatly improve the computational complexity of KPCA. In the case of large-scale or online dataset, the computational superiority of our approach is remarkable. Experimental results on synthetic and real datasets demonstrate that TP-IKPCA can significantly improve the time complexity of KPCA while preserving a high accuracy as standard KPCA. In comparison with two incremental KPCA algorithms, TP-IKPCA also illustrates superiority in terms of training and testing speed.

TP-IKPCA can be utilized in any application where KPCA needs to be conducted, especially when training data is of large scale, or can only be collected one by one, where the conventional batch-based KPCA cannot be applied. The idea of this study can be extended to other kernel-based methods, such as Kernel Fisher discriminant analysis (KFDA), Kernel independent component analysis (KICA), and so on.

Appendix

A. The Proof of Theorem 1

Let $\{\boldsymbol{\beta}_j\}_{j=1}^{r}$ denote an orthonormal basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N}$. Equation (1) can be written as

Table 3: Training and testing time (sec) on digit "0" when increasing the number of training samples from 500 to 5000.

N		500	1000	1500	2000	3000	4000	5000
r		242	366	453	514	605	670	735
N/r		**2.07**	**2.73**	**3.31**	**3.89**	**4.96**	**5.97**	**6.80**
training stage	KPCA	1.75 ± 0.055	6.98 ± 0.066	18.72 ± 0.140	35.46 ± 0.185	79.97 ± 0.570	154.65 ± 0.687	255.01 ± 2.083
	TP-IKPCA	1.89 ± 0.019	5.34 ± 0.064	9.86 ± 0.085	14.92 ± 0.078	25.40 ± 0.015	37.46 ± 0.031	50.43 ± 0.411
	ratio	**0.93**	**1.31**	**1.90**	**2.38**	**3.15**	**4.13**	**5.06**
testing stage	KPCA	0.358 ± 0.022	0.650 ± 0.013	1.131 ± 0.018	1.600 ± 0.026	2.406 ± 0.070	3.657 ± 0.170	5.234 ± 0.293
	TP-IKPCA	0.181 ± 0.012	0.266 ± 0.014	0.323 ± 0.017	0.371 ± 0.016	0.429 ± 0.429	0.465 ± 0.011	0.511 ± 0.010
	ratio	**1.98**	**2.44**	**4.06**	**4.31**	**5.61**	**8.02**	**10.24**

$$\boldsymbol{Y} = \boldsymbol{B}^T \boldsymbol{\varphi}(\boldsymbol{X}), \tag{A.1}$$

where $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N]$ is the projection matrix, $\mathbf{B} = [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r]$ denotes the orthonormal basis matrix, and $\boldsymbol{\varphi}(\boldsymbol{X}) = [\boldsymbol{\varphi}(\boldsymbol{x}_1), \boldsymbol{\varphi}(\boldsymbol{x}_2), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_N)]$ is the mapped sample matrix. The covariance matrix $\boldsymbol{C}$ of $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^N$ (see (2)) can be expressed using the following formula:

$$\boldsymbol{C} = \boldsymbol{\varphi}(\boldsymbol{X}) \boldsymbol{H}_N \boldsymbol{H}_N^T \boldsymbol{\varphi}(\boldsymbol{X})^T, \tag{A.2}$$

where $\boldsymbol{H}_N$ is a $N \times N$ matrix and its element h_{ij} can be written as

$$h_{ij} = \begin{cases} \dfrac{(N-1)}{N} & i = j \\ \dfrac{-1}{N} & i \neq j \end{cases} \quad (1 \leq i, j \leq N). \tag{A.3}$$

Combining (A.1) and (A.2), we have

$$\boldsymbol{C} = \boldsymbol{B}\left(\boldsymbol{Y}\boldsymbol{H}_N\boldsymbol{H}_N^T\boldsymbol{Y}^T\right)\boldsymbol{B}^T. \tag{A.4}$$

Let $\overline{\boldsymbol{C}} = \boldsymbol{Y}\boldsymbol{H}_N\boldsymbol{H}_N^T\boldsymbol{Y}^T$, then $\overline{\boldsymbol{C}}$ is the covariance matrix of $\boldsymbol{Y}$. We have

$$\boldsymbol{C} = \boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{B}^{\boldsymbol{T}}. \tag{A.5}$$

For the relationship of the eigenvalues and the corresponding eigenvector between $\boldsymbol{C}$ and $\overline{\boldsymbol{C}}$, we give a derivation as follows.

Lemma A.1. *Let $\lambda_l \neq 0$ be the l-th nonzero eigenvalue of $\boldsymbol{C}$ and $\boldsymbol{v}_l$ be the eigenvector, then $\boldsymbol{u}_l = \boldsymbol{B}^T\boldsymbol{v}_l$ is the eigenvector of $\overline{\boldsymbol{C}}$ and its eigenvalue is λ_l. Scilicet, $\overline{\boldsymbol{C}}\boldsymbol{u}_l = \lambda_l\boldsymbol{u}_l$.*

Proof. Based on the above definitions, we have $\boldsymbol{C}\boldsymbol{v}_l = \lambda_l\boldsymbol{v}_l$ and$\boldsymbol{v}_l = \boldsymbol{B}\boldsymbol{u}_l$. On the other hand, $\boldsymbol{C} = \boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{B}^T$ is established. So, $\boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{u}_l = \lambda_l\boldsymbol{B}\boldsymbol{u}_l$, thus, $\boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{u}_l = \lambda_l\boldsymbol{B}\boldsymbol{u}_l$. Hence, $\boldsymbol{B}^T\boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{u}_l = \lambda_l\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{u}_l$, Scilicet, $\overline{\boldsymbol{C}}\boldsymbol{u}_l = \lambda_l\boldsymbol{u}_l$. □

Lemma A.2. *Let $\lambda_l \neq 0$ be the l-th nonzero eigenvalue of $\overline{\boldsymbol{C}}$ and $\boldsymbol{u}_l$ be the corresponding eigenvector, then $\boldsymbol{v}_l = \boldsymbol{B}\boldsymbol{u}_l$ is the eigenvector of $\boldsymbol{C}$ and its eigenvalue is λ_l. Scilicet, $\boldsymbol{C}\boldsymbol{v}_l = \lambda_l\boldsymbol{v}_l$.*

Proof.

$$\boldsymbol{C}\boldsymbol{v}^l = \boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{u}_l = \boldsymbol{B}\overline{\boldsymbol{C}}\boldsymbol{u}_l = \boldsymbol{B}\lambda_l\boldsymbol{u}_l = \lambda_l\boldsymbol{v}_l. \tag{A.6}$$

Based on Lemmas A.1 and A.2, we know the nonzero eigenvalue of $\boldsymbol{C}$ is the same with that of $\overline{\boldsymbol{C}}$ and the corresponding eigenvector satisfies $\boldsymbol{v}_l = \boldsymbol{B}\boldsymbol{u}_l$. Hence, the eigendecomposition of $\boldsymbol{C}$ can be converted into the corresponding process of $\overline{\boldsymbol{C}}$. And because $\boldsymbol{v}_l/\|\boldsymbol{v}_l\| = \boldsymbol{B}\boldsymbol{u}_l/\|\boldsymbol{B}\boldsymbol{u}_l\| = \boldsymbol{B}\boldsymbol{u}_l/\|\boldsymbol{u}_l\|$, so (8) still holds after the eigenvector is unitized. So, Theorem 1 is proven. □

B. The Proof of Lemma 2

Let $\boldsymbol{\Gamma} = [\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2, \ldots, \boldsymbol{\gamma}_r]$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_r)$ be the corresponding matrix which are, respectively, from the eigenvector and the eigenvalue of the kernel matrix $\boldsymbol{K}_{rr} = [k(\boldsymbol{x}_{bs}, \boldsymbol{x}_{bt})]_{1\leq s,t\leq r}$. We have $\boldsymbol{\Gamma}^T\boldsymbol{K}_{rr}\boldsymbol{\Gamma} = \boldsymbol{\Lambda}$. Let $\boldsymbol{B} = [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r]$ and $\boldsymbol{\Delta} = \mathrm{diag}(1/\sqrt{\varepsilon_1}, 1/\sqrt{\varepsilon_2}, \ldots, 1/\sqrt{\varepsilon_r})$; we have $\boldsymbol{D} = \boldsymbol{\Gamma}\boldsymbol{\Delta}$. Thus $\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{D}^T\boldsymbol{K}_{rr}\boldsymbol{D} = \boldsymbol{\Delta}^T\boldsymbol{\Gamma}^T\boldsymbol{K}_{rr}\boldsymbol{\Gamma}\boldsymbol{\Delta} = \boldsymbol{\Delta}^T\boldsymbol{\Lambda}\boldsymbol{\Delta} = \boldsymbol{I}$. So, Lemma 2 is proven.

C. The Proof of Lemma 3

If $\{\boldsymbol{\beta}_j\}_{j=1}^r$ is the orthonormal basis of the subspace spanned by $\{\boldsymbol{\varphi}(\boldsymbol{x}_i)\}_{i=1}^{N+1}$, then $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1}) = [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r][\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r]^T\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$. Based (9), we have

$$\begin{aligned} &\boldsymbol{\varphi}(\boldsymbol{x}_{N+1}) = [\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})] \\ &\quad \cdot \boldsymbol{D}\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)} \Longleftrightarrow \\ &\left\|\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})\right. \\ &\quad \left. - [\boldsymbol{\varphi}(\boldsymbol{x}_{b1}), \boldsymbol{\varphi}(\boldsymbol{x}_{b2}), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_{br})]\, \boldsymbol{D}\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)}\right\| \\ &\quad = 0 \Longleftrightarrow \\ &k_{N+1} - \boldsymbol{k}_{b(N+1)}^T\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)} = 0. \end{aligned} \tag{C.1}$$

So, conclusion (1) in Lemma 3 is proven. Subsequently, if $\delta = k_{N+1} - \boldsymbol{k}_{b(N+1)}^T\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)} \neq 0$, this means $\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})$ cannot be linearly expressed by $\{\boldsymbol{\beta}_j\}_{j=1}^r$. Based the Gram-Schmidt orthogonalization process, $\boldsymbol{\beta}_{r+1}$ can be determined using the following formula:

$$\begin{aligned} &\boldsymbol{\beta}_{r+1} \\ &= \frac{\boldsymbol{\varphi}(\boldsymbol{x}_{N+1}) - [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r][\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r]^T\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})}{\left\|\boldsymbol{\varphi}(\boldsymbol{x}_{N+1}) - [\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r][\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_r]^T\boldsymbol{\varphi}(\boldsymbol{x}_{N+1})\right\|}. \end{aligned} \tag{C.2}$$

Combined with (9) and kernel function, (C.2) can be written using the following formula:

$$\begin{aligned} \boldsymbol{\beta}_{r+1} &= [\boldsymbol{\varphi}(\boldsymbol{x}_1), \boldsymbol{\varphi}(\boldsymbol{x}_2), \ldots, \boldsymbol{\varphi}(\boldsymbol{x}_N), \boldsymbol{\varphi}(\boldsymbol{x}_{N+1})] \\ &\quad \cdot \begin{bmatrix} \dfrac{-\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{k}_{b(N+1)}}{\sqrt{|\delta|}} \\ 1 \\ \dfrac{}{\sqrt{|\delta|}} \end{bmatrix}. \end{aligned} \tag{C.3}$$

Then, we can obtain (11) and (12). So Lemma 2 is proven.

Data Availability

In our manuscript, we used two datasets to support the findings of our study. One dataset is the synthetic toy data, which can be generated by the following way: $\boldsymbol{y} = (\boldsymbol{x}\ 2\ 2 + 1) + 0.2\boldsymbol{\xi}$, $\boldsymbol{\xi} \sim \boldsymbol{N}(0,1)$, $\boldsymbol{x} \sim \boldsymbol{U}[0,1]$ (1) where $\boldsymbol{N}(0,1)$ denotes the standard Gaussian distribution and $\boldsymbol{U}[0,1]$ is the uniform distribution in $[0,1]$. The second data set is the MNIST database of handwritten digits, which are available on this site: http://yann.lecun.com/exdb/mnist.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (Grants nos. 61773244, 61373079, and 61572344), National Institutes of Health in USA (AG041721, MH107815, EB006733, EB008374, and EB009634), and Provincial Natural Science Foundation of Shanxi in China (2018JM4018).

References

[1] I. T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, New York, NY, USA, 1986.

[2] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[3] B. Chen, J. Yang, B. Jeon, and X. Zhang, "Kernel quaternion principal component analysis and its application in RGB-D object recognition," *Neurocomputing*, vol. 266, pp. 293–303, 2017.

[4] X. Deng and L. Wang, "Modified kernel principal component analysis using double-weighted local outlier factor and its application to nonlinear process monitoring," *ISA Transactions®*, vol. 72, pp. 218–228, 2018.

[5] Y. Yang, W. Sheng, Y. Han, and X. Ma, "Multi-beam pattern synthesis algorithm based on kernel principal component analysis and semi-definite relaxation," *IET Communications*, vol. 12, no. 1, pp. 82–95, 2018.

[6] K. I. Kim, M. O. Franz, and B. Schölkopf, "Iterative kernel principal component analysis for image modeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 9, pp. 1351–1366, 2005.

[7] A. R. Teixeira, A. M. Tomé, K. Stadlthanner, and E. W. Lang, "KPCA denoising and the pre-image problem revisited," *Digital Signal Processing*, vol. 18, no. 4, pp. 568–580, 2008.

[8] W. Soh, H. Kim, and B.-J. Yum, "Application of kernel principal component analysis to multi-characteristic parameter design problems," *Annals of Operations Research*, vol. 263, no. 1-2, pp. 69–91, 2018.

[9] R. Rosipal and M. Girolami, "An expectation-maximization approach to nonlinear component analysis," *Neural Computation*, vol. 13, no. 3, pp. 505–510, 2001.

[10] G. Simon, N. S. Nicol, and S. V. N. Vishwanathan, "Fast iterative kernel principal component analysis," *Journal of Machine Learning Research (JMLR)*, vol. 8, no. 4, pp. 1893–1918, 2007.

[11] W. Zheng, C. Zou, and L. Zhao, "An improved algorithm for kernel principal component analysis," *Neural Processing Letters*, vol. 22, no. 1, pp. 49–56, 2005.

[12] F. Vojtěch and H. Václav, "Greedy algorithm for a training set reduction in the kernel methods," in *Proceedings of the 10th International Conference on Computer Analysis of Image and Patterns*, vol. 2756 of *Lecture Notes in Comput. Sci.*, pp. 426–433, Springer, Groningen, Netherlands, August 2003.

[13] F. Vojtech, *Optimization Algorithms for Kernel Methods [Ph.D. Dissertation]*, Center for Machine Perception, Czech Technical University, Prague, Czech Republic, 2005.

[14] T. Chin and D. Suter, "Incremental kernel PCA for efficient non-linear feature extraction," in *Proceedings of the 17th British Machine Vision Conference*, pp. 4–7, Edinburgh, Scotland, September 2006.

[15] T.-J. Chin and D. Suter, "Incremental kernel principal component analysis," *IEEE Transactions on Image Processing*, vol. 16, no. 6, pp. 1662–1674, 2007.

[16] B. J. Kim and I. K. Kim, "Incremental nonlinear PCA for classification," in *Proceedings of the European Conference on Knowledge Discovery in Databases (PKDD)*, vol. 3202 of *Lecture Notes in Computer Science*, pp. 291–300, Springer, 2004.

[17] B.-J. Kim, "Active visual learning and recognition using incremental kernel PCA," in *Proceedings of the 18th Australian Joint Conference on Advances in Artificial Intelligence AI'05*, vol. 3809 of *Lecture Notes in Comput. Sci.*, pp. 585–592, Springer, 2005.

[18] P. M. Hall, D. Marshall, and R. R. Martin, "Incremental eigenanalysis for classification," in *Proceedings of the British Machine Vision Conference*, pp. 286–295, 1998.

[19] S. Kimura, S. Ozawa, and S. Abe, "Incremental Kernel PCA for online learning of feature space," in *Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation*, vol. 1, pp. 595–600, Vienna, Austria, November 2005.

[20] Y. Takeuchi, S. Ozawa, and S. Abe, "An efficient incremental kernel principal component analysis for online feature selection," in *Proceedings of the 2007 International Joint Conference on Neural Networks*, pp. 2346–2351, Orlando, FL, USA, August 2007.

[21] O. Seiichi, Y. Takeuchi, and A. Shigeo, "A fast incremental kernel principal component analysis for online feature extraction," in *Proceedings of the Pacific Rim International Conference on Trends in Artificial Intelligence*, vol. 6230 of *Lecture Notes in Computer Science*, pp. 487–497, Springer, 2010.

[22] T. Takaomi and O. Seiichi, "A fast incremental kernel principal component analysis for learning stream of data chunks," in *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN 2011 - San Jose)*, pp. 2881–2888, San Jose, CA, USA, July 2011.

[23] A. A. Joseph and S. Ozawa, "A fast incremental kernel principal component analysis for data streams," in *Proceedings of the 2014 International Joint Conference on Neural Networks (IJCNN)*, Beijing, China, July 2014.

[24] A. A. Joseph, T. Tokumoto, and S. Ozawa, "Online feature extraction based on accelerated kernel principal component analysis for data stream," *Evolving Systems*, vol. 7, no. 1, pp. 15–27, 2016.

[25] H. Fredrik and N. Paul, "Incremental kernel PCA and the Nyström method," 2018, https://arxiv.org/abs/1802.00043.

[26] G. Baudat and F. Anouar, "Kernel-based methods and function approximation," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN'01)*, pp. 1244–1249, Washington, DC, USA, July 2001.

[27] H. Zhao, P. C. Yuen, and J. T. Kwok, "A novel incremental principal component analysis and its application for face recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 4, pp. 873–886, 2006.

[28] J. Weng, Y. Zhang, and W. Hwang, "Candid covariance-free incremental principal component analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 1034–1040, 2003.

[29] S. Nicole, "Feedforward neural networks for principal components extraction," *Computational Statistics & Data Analysis*, vol. 33, no. 4, pp. 425–437, 2000.

[30] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, no. 6, pp. 459–473, 1989.

[31] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, 1982.

[32] Y. Li, "On incremental and robust subspace learning," *Pattern Recognition*, vol. 37, no. 7, pp. 1509–1518, 2004.

[33] M. Artac, M. Jogan, and A. Leonardis, "Incremental PCA for on-line visual learning and recognition," in *Proceedings of the 16th International Conference on Pattern Recognition*, pp. 781–784, Quebec City, Canada, 2002.

[34] O. Seiichi, P. Shaoning, and K. Nikola, "A modified incremental principal component analysis for on-line learning of feature space and classifier," in *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence*, pp. 231–240, Auckland, New Zealand, 2004.

[35] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, NY, USA, 2013.

[36] S. Ling, X. Cheng, and T. Jiang, "An algorithm for coneigenvalues and coneigenvectors of quaternion matrices," *Advances in Applied Clifford Algebras (AACA)*, vol. 25, no. 2, pp. 377–384, 2015.

[37] C. Han, Y. Wang, and G. He, "On the convergence of asynchronous parallel algorithm for large-scale linearly constrained minimization problem," *Applied Mathematics and Computation*, vol. 211, no. 2, pp. 434–441, 2009.

[38] S. B. Mike, B. Scholkopf, and A. J. Smola, "Kernel PCA and denoising in feature space," in *Advances in Neural Information Processing System*, pp. 524–536, MTI press, Cambridge, UK, 1999.